

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平8-69417

(43)公開日 平成8年(1996)3月12日

(51) Int.Cl.
G 0 6 F 12/12

識別記号 庁内整理番号
A 7623-5B

F I

技術表示箇所

審査請求 未請求 請求項の数 2 OL (全 10 頁)

(21)出願番号 特願平6-203253

(22)出願日 平成6年(1994)8月29日

(71)出願人 000001889

三洋電機株式会社

大阪府守口市京阪本通2丁目5番5号

(72)発明者 甲村 康人

大阪府守口市京阪本通2丁目5番5号 三
洋電機株式会社内

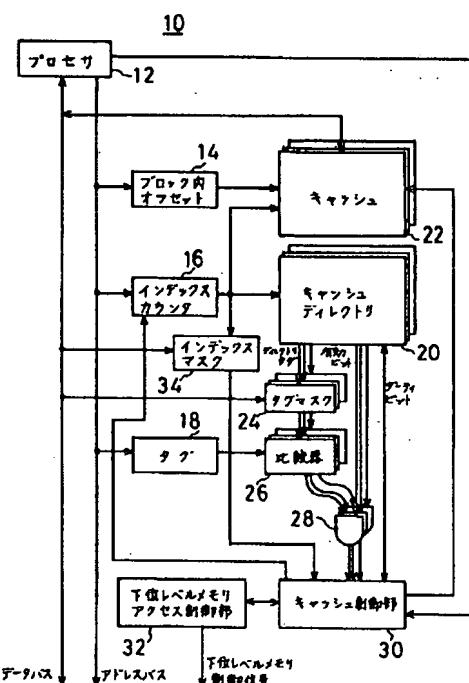
(74)代理人 弁理士 山田 義人

(54)【発明の名称】 計算機システム

(57)【要約】

【構成】 プロセサ12の要求に従い指定されたアドレス範囲に対応するキャッシュブロックが、キャッシュディレクトリ20からのディレクトリタグとプロセサ12からのタグとを比較器26で比較することによって検出する。このとき、ディレクトリタグをタグマスク22によってマスクすることによって効率的にキャッシュブロックを検出できる。また、プロセサ12から出力されたインデックスの値をインデックスカウンタ16によってインクリメントし、インデックスマスク34によってアドレス範囲に応じてインデックスカウンタ16の上限を検出する。インデックスカウンタ16の上限が検出されるまでの間、プロセサ12によって指定されたアドレス範囲に対応するキャッシュブロックのダーティビットをクリアする。

【効果】 下位レベルのメモリブロックに対する不要の書き出しを防ぐことができ、計算機システムの性能が向上する。



1

【特許請求の範囲】

【請求項1】下位メモリと前記下位メモリの内容のコピーをブロック単位でもつキャッシュとを含み、前記キャッシュはプロセサからのライトアクセス要求に対してはライトバック方式によってこれを処理する計算機システムにおいて、

前記プロセサからの要求に応じて、指定されたメモリ領域に対応するキャッシュブロックのダーティビットを強制的にクリアするクリア手段を備えることを特徴とする、計算機システム。

【請求項2】前記クリア手段は、前記プロセサによって指定されたアドレス範囲に対応するキャッシュブロックのダーティビットをクリアするキャッシュ制御手段、前記アドレス範囲に対応する前記キャッシュブロックをタグマスクを用いて検出するブロック検出手段、前記プロセサから出力されるインデックスの値をインクリメントするインデックスカウンタ、および前記アドレス範囲に応じて前記インデックスカウンタの上限を検出するインデックス検出手段を備え、

前記インデックス検出手段で前記インデックスカウンタの上限が検出されるまでの間前記アドレス範囲に対応する前記キャッシュブロックのダーティビットを前記キャッシュ制御手段によってクリアする、請求項1記載の計算機システム。

【発明の詳細な説明】

【0001】

【産業上の利用分野】この発明は計算機システムに関し、特にたとえばキャッシュメモリの管理方式にライトバック方式を採用する、計算機システムに関する。

【0002】

【従来の技術】近年の高性能の計算機システムには、キャッシュと呼ばれる上位レベルの高速メモリに下位レベルのメモリの内容の一部をコピーしてもつことによってシステムの性能を向上させる機構が、広く用いられている。キャッシュの管理方式の1つは、プロセサからメモリシステムへの書き込みをキャッシュおよび下位メモリの両方に対して行い、キャッシュの内容を常に下位レベルのメモリの内容と一緒にさせておくライトスルーワー方式であり、別の1つは、プロセサからメモリシステムへの書き込みをキャッシュに対してのみ行い、データが変更されたキャッシュの内容はリプレースの対象になった時点で下位レベルのメモリに書き出されるライトバック方式である。

【0003】ライトバック方式では、キャッシュブロックの内容がプロセサによって変更されたため下位レベルのメモリの内容との一貫性を失っている状態をダーティであるといい、キャッシュブロックのそれぞれについて、ダーティか否かを示すためのダーティビットと呼ばれる状態ビットが設けられている。たとえば、従来のライトバック方式かつセットアソシエティプキャッシュ方

2

式の計算機システム1は図5のように構成される。

【0004】図5を参照して、従来の計算機システム1において、プロセサ2から階層メモリに対するリードアクセス要求が出されたとする。プロセサ2から出力されたアドレスはブロック内オフセット、インデックスおよびタグに分割して保持され、インデックスで示されるキャッシュディレクトリ3のエントリがキャッシュ4の連想度の数だけ並列にアクセスされ、キャッシュディレクトリ3からディレクトリタグ、有効ビットおよびダーティビットが出力される。ディレクトリタグとアドレスのタグとは比較器5によって比較され、これらが一致し、かつ有効ビットが真であるディレクトリエントリが存在すれば、リードアクセスはキャッシュ4にヒットしたことになり、キャッシュ4がインデックスおよびプロセサ内オフセットによってアクセスされ、データがプロセサ2に返される。

【0005】もし、インデックスで示されるセット内の全てのエントリについて、有効ビットが偽であるかまたはディレクトリタグがアドレスのタグと一致しないのであれば、キャッシュミスが発生したことになる。この場合、キャッシュ制御部6はキャッシュ4中のどのキャッシュブロックを今回アクセスのあったメモリブロックの内容でリプレースするかを選択する必要がある。

【0006】リプレースの対象となったキャッシュブロックの有効ビットが偽であるか、ダーティビットが偽であるなら、そのキャッシュブロックの内容は不要あるいは下位レベルのメモリ中に同じ内容のものが存在するから、捨ててしまってかまわない。このときは、下位レベルメモリアクセス制御部7は、プロセサ2からのアクセス要求のあったアドレスを含むメモリブロックのリード要求（下位レベルメモリ制御信号）を下位レベルのメモリに対して発生する。そして、キャッシュ4のいずれかのセットが選択され、そのインデックスによって示されるエントリにこのメモリブロックの内容が転送される。また、そのメモリブロックのうちプロセサ2から要求のあったデータワードがプロセサ2にも返される。さらに、選択されたエントリのタグ部には今回アクセスされたアドレスのタグ部が設定され、有効ビットは真に、ダーティビットは偽に設定される。

【0007】一方、リプレースの対象となったキャッシュブロックの有効ビットが真であり、ダーティビットが真であるなら、キャッシュ制御部6はこのキャッシュブロックの内容をまず下位レベルのメモリに書き出して、その後にキャッシュブロックに新たな内容を読み込み上述の操作を行う必要がある。ところで、現在の典型的なプログラム、特に大規模な処理を行うプログラムにおいては、必要になった時点で生成されて不要になれば消去されるような動的な変数が多用される。このような動的変数の生成から消去までの期間を変数の生存期間と呼ぶ。動的変数のために用いられるメモリ領域は、その動

3

的変数が最初に必要とされる時点でメモリの未使用領域から割り当てられ、その動的変数が不要になった時点で割り当てられていたメモリ領域は未使用領域に返却される。

【0008】動的変数の種類の1つは、プログラムの変数コール／リターンに伴って生成／消去されるものであり、関数の局所変数と呼ばれる。この種の動的変数はスタック領域と呼ばれるメモリ領域に割り当てられる。現在の多くのプロセサにおいては、スタック領域を管理するためのハードウェア機構を備えている。また、個々の関数呼び出しに対応して確保されるスタック領域をスタックフレームと呼ぶ。スタックフレームは、関数の局所変数、関数にわたされる引数に関する情報、および関数からのリターンアドレスなどを含む。

【0009】動的変数の別の種類は、その生存期間がプログラムの関数呼び出しと関連づけることができないものである。このような動的変数はヒープ領域と呼ばれるメモリ領域に割り当てられる。通常、この種の動的変数に対するメモリ領域の割り当ての管理は、ソフトウェアによって行われる。

【0010】

【発明が解決しようとする課題】ここで、図5に示すライトパック方式のキャッシュ4をもつ既存の計算機システム1について考える。キャッシュ4上に動的変数に対応するキャッシュブロックが存在し、なおかつこの動的変数の生存期間が終了した時点で、そのキャッシュブロックがダーティである場合を考える。この時点で、このキャッシュブロックが対応する下位レベルのメモリ領域は、ヒープ領域あるいはスタッ�領域であり、いずれの動的変数も割り当てられていない状態である。さらに処理が進み、このキャッシュブロックに対応するメモリ領域に新たな動的変数が割り当てられるより以前に、このキャッシュブロックがリプレースの対象になったとする。このキャッシュブロックはダーティであるため、下位レベルのメモリに内容の書き出しが行われる。

【0011】しかし、このキャッシュブロックの内容は、生存期間が終了した動的変数の内容であり、もはやプログラムによって必要とされない。すなわち、これまでの計算機システムは、生存期間の終了した動的変数がキャッシュブロック上にダーティブロックとして存在しかつそのキャッシュブロックがリプレースの対象となつたときに、下位レベルのメモリにキャッシュブロックの内容を書き出すという不要な操作を行っていた。

【0012】それゆえに、この発明の主たる目的は、不要な操作を防ぎ性能を向上することができる、計算機システムを提供することである。

【0013】

【課題を解決するための手段】この発明は、下位メモリと下位メモリの内容のコピーをブロック単位でもつかッショとを含み、キャッシュはプロセサからのライトア

4

クセス要求に対してはライトパック方式によってこれを処理する計算機システムにおいて、プロセサからの要求に応じて、指定されたメモリ領域に対応するキャッシュブロックのダーティビットを強制的にクリアするクリア手段を備えることを特徴とする、計算機システムである。

【0014】

【作用】プロセサからの要求に従い、指定されたアドレス範囲に対応するキャッシュブロックをブロック検出手段でタグマスクを用いて効率よく検出する。また、インデックスカウンタによって、プロセサから出力されたインデックスの値をインクリメントし、インデックス検出手段で、指定されたアドレス範囲に応じてインデックスカウンタの上限を検出する。そして、インデックスカウンタの上限が検出されるまでの間、プロセサによって指定されたアドレス範囲に対応するキャッシュブロックのダーティビットを、キャッシュ制御手段によって強制的にクリアする。

【0015】ここで、強制的なダーティビットのクリアは、キャッシュブロックの内容が下位レベルのメモリブロックとの同一性を失っているにも拘わらず、その情報が計算機システムから失われてしまう、すなわち、計算機システムが一貫性を失ってしまうことを意味する。このことが正しいプログラムの実行に影響を与えないことを以下に示す。

【0016】ここでは、動的変数の生存期間が終了した場合を例に説明する。一般に、動的変数が割り当てられているメモリ領域は、プログラム実行時に決定され、どのメモリ領域が割り当てられているか、さらに、割り当てられた領域に偶然格納されている値がどのようなものがあるか、プログラマーが事前に知ることはできない。すなわち、動的変数の初期値はプログラマーにとっては非決定的であると考えられている。したがって、正しいプログラムは動的変数の初期値に依存しないようにコーディングされている。

【0017】ここで、アドレスaに割り当てられた動的変数の生存期間が終了し、アドレスaに対応するキャッシュブロックcのダーティビットが強制的にクリアされたとする。この時点以後、キャッシュブロックcに対する最初の操作は、読み出し、書き込み、リプレースのいずれかである。キャッシュブロックcに対して読み出しが実行される場合は、アドレスaに新たな動的変数が割り当てられ、それが参照される場合のみである。しかし、上述のように動的変数の初期値に依存するプログラムは正しいプログラムとはいえないため、この場合を考慮する必要はない。

【0018】キャッシュブロックcに対して書き込みが実行される場合は、アドレスaに新たな動的変数が割り当てられ、それに対する代入が行われた場合である。このとき、キャッシュブロックcのダーティビットは真と

なる。これにより、計算機システムの一貫性は回復され、先に実行されたダーティビットのクリア操作が、これ以降の実行に影響を及ぼすことはない。

【0019】キャッシュブロックcに対してリプレースが実行された場合、プログラムから見たアドレスaの内容は、下位レベルのメモリブロックの内容となる。これはプロセサが最後（リプレース直前）にアドレスaに対して書き込んだ内容、すなわちキャッシュブロックcの内容とは異なる。しかし、アドレスaは動的変数のための領域すなわちスタック領域あるいはヒープ領域であり、アドレスaには動的変数が割り当てられていない状態である。将来、プロセサがアドレスaのメモリ領域を利用する場合は、新たな動的変数がアドレスaに割り当てられることを意味するが、上述のように正しいプログラムは動的変数の初期値に依存しないように設計されているため、アドレスaのメモリブロックに格納されている値はどのような値でもプログラムの実行に影響を及ぼすことはない。したがって、先に実行されたダーティビットクリア操作が、これ以降の実行に影響を及ぼすことはない。

【0020】

【発明の効果】この発明によれば、ダーティなキャッシュブロックがリプレースの対象となった場合に、キャッシュブロックのダーティビットを強制的にクリアすることによって、正しいプログラムの実行に影響を与えることなく下位レベルのメモリブロックに対する不要な書き出しを防ぐことができ、ひいては計算機システムの性能を向上させることができる。

【0021】この発明の上述の目的、その他の目的、特徴および利点は、図面を参照して行う以下の実施例の詳細な説明から一層明らかとなろう。

【0022】

【実施例】図1を参照して、この実施例の計算機システム10は、たとえば、ライトバック方式でありますかセットアソシエイティブキャッシュ方式に構成されたものである。計算機システム10は、プロセサ12を含む。プロセサ12から階層メモリシステムに対するアクセス要求のあったアドレスは、ブロック内オフセット、インデックスおよびタグに分割され、それぞれブロック内オフセット用レジスタ14、インデックスカウンタ16およびタグ用のレジスタ18に保持される。そして、インデックスで示されるキャッシュディレクトリ20のエントリがキャッシュ22の連想度の数だけ並列にアクセスされ、キャッシュディレクトリ20からディレクトリタグ、有効ビットおよびダーティビットが出力される。ディレクトリタグはタグマスク24によってマスクされた後、レジスタ18からのアドレスのタグと比較器26によって比較され、これらが一致し、かつ有効ビットが真であるディレクトリエントリすなわちキャッシュブロックが存在すれば、リードアクセスはキャッシュ22にヒ

ットしたことになり、ANDゲート2.8を介してその旨の信号がキャッシュ制御部30に与えられる。すると、キャッシュ制御部30によって、キャッシュ22がインデックスおよびプロセサ内オフセットに基づいてアクセスされ、データがプロセサ12に返される。

【0023】もし、インデックスで示されるセット内の全てのエントリについて、有効ビットが偽であるかまたはディレクトリタグがアドレスのタグと一致しないのであれば、キャッシュミスが発生したことになる。この場合、図5に示す従来技術と同様、キャッシュ制御部30は、キャッシュ22中のどのキャッシュブロックを今回アクセスのあったメモリブロックの内容でリプレースするかを選択する必要がある。

【0024】すなわち、リプレースの対象となったキャッシュブロックの有効ビットが偽であるか、ダーティビットが偽であるなら、そのキャッシュブロックの内容は不要あるいは下位レベルのメモリ中に同じ内容のものが存在するから、捨ててしまってかまわない。このときは、下位レベルメモリアクセス制御部32は、プロセサ12からのアクセス要求のあったアドレスを含むメモリブロックのリード要求（下位レベルメモリ制御信号）を下位レベルのメモリ（図示せず）に対して発生する。そして、キャッシュ22のいずれかのセットが選択されて、そのインデックスによって示されるエントリにこのメモリブロックの内容が転送される。また、そのメモリブロックのうちプロセサ12から要求のあったデータカードがプロセサ12にも返される。さらに、選択されたエントリのタグ部には今回アクセスされたアドレスのタグ部が設定され、有効ビットは真に、ダーティビットは偽に設定される。

【0025】また、リプレースの対象となったキャッシュブロックの有効ビットが真であり、ダーティビットが真であるなら、キャッシュ制御部30はこのキャッシュブロックの内容をまず下位レベルのメモリに書き出して、その後にキャッシュブロックに新たな内容を読み込み上述の操作を行う。このように、計算機システム10は、図5に示す従来の計算機システム1の有する機能を備えているが、計算機システム10ではさらに以下の点に注目すべきである。

【0026】すなわち、計算機システム10では、プロセサ12からの要求に従い、指定されたアドレス範囲に応するキャッシュブロックのダーティビットをクリアする機能をもつよう改良されたキャッシュ制御部30、指定されたアドレス範囲に対応するキャッシュブロックを効率よく見出すために、ディレクトリタグをマスクするタグマスク24、インデックスの値をインクリメントするインデックスカウンタ16、およびインデックスカウンタ16の上限を与えるインデックスマスク34を備えることである。

【0027】タグマスク24は、たとえば図2に示すよ

7

うに構成される。図2に示すタグマスク24は、ビット毎のNOT回路24aを含み、データバスに現れるmask-tagをNOT回路24aによってビット毎に否定演算し、得られた～mask-tagをタグマスクレジスタ24bに与え、後述する図4のアルゴリズムの実行中に～mask-tagはタグマスクレジスタ22bで保持される。また、キャッシュディレクトリ20によって生成されるディレクトリタグの値とタグマスクレジスタ24bの内容とのビット毎の論理積がAND回路24cによって生成され、得られた結果が比較器26へ転送される。このようなタグマスク24を用いかつ後述の制約条件を満たすことによって、検査すべきキャッシュブロックの数を最小限にでき、その結果、キャッシュブロックの検出を効率的に行える。

【0028】また、インデックスマスク34は、たとえば図3に示すように構成される。図3に示すインデックスマスク34は、インデックスマスクレジスタ34aを含む。インデックスマスクレジスタ34aは、図4に示すアルゴリズムを実行する間、mask-idxを保持しておく。インデックスマスクレジスタ34aからの値とインデックスカウンタ16によって生成されるインデックス値とのビット毎の論理積がAND回路34bによって生成される。さらに、AND回路34bによって生成される値とインデックスマスクレジスタ34aに保持される値が比較器34cによって比較され、それらの値が等しいときには、比較器34cからキャッシュ制御部30に図4に示すアルゴリズムの終了が通知される。すなわち、インデックスマスク34によって、プロセサ12から指示されたアドレス範囲に対応してインデックスカウンタ16の上限を検出している。

【0029】図1に戻って、さらに、プロセサ12は、ユーザプログラムにて実行可能な、特定のアドレス範囲に対応するキャッシュブロックのダーティビットをクリアするための命令

clean-cache addr mask

をもつ。この命令は、たとえば、動的変数の生存期間が終了した時点でプロセサ12から出力され、キャッシュ制御部30に入力される。またこの命令は、2のべき乗の値をとるアドレスaddrと2のべき乗-1の値をとるmaskとをオペランドとし、addrからaddr+maskまでのアドレスに対応するキャッシュブロックを全て見出し、そのキャッシュブロックのダーティビットをクリアする操作をキャッシュ制御部30に指示する。ただし、mask+1はキャッシュ22のブロックサイズ以上でなければならず、addrとmaskとのビット毎の論理積をとったものは0に等しいという制約条件を満たすものとする。

【0030】キャッシュ制御部30は、上述のプロセサ12からの命令の要求に応じて、図4に示す手順に従ってキャッシュブロックのサーチおよびダーティビットクリアの操作を行う。ここで、**clean-cache** 命令のオペラ

8

ンドであるaddrはアドレスバスに、maskはデータバスにそれぞれプロセサ12から出力されるものとする。図4に示す動作において、まず、**clean-cache** 命令に与えられたaddrおよびmaskのそれぞれのタグ部をaddr-tagおよびmask-tagとし、addrおよびmaskのそれぞれのインデックス部をaddr-idxおよびmask-idxとする。

【0031】そして、ステップS1において、addr-tagをタグ用のレジスタ18に、～mask-tagをタグマスクレジスタ24bに、addr-idxをインデックスカウンタ16に、mask-idxをインデックスマスク34に、それぞれ保持する。その後、キャッシュ22の各セットについて並列にステップS3およびS5をそれぞれ実行する。ステップS3において、ディレクトリタグ&タグマスクレジスタ24bの出力(～mask-tag) = タグ用のレジスタ18の出力(addr-tag)であるか否かが判断される。これは、タグマスク24および比較器26で行われる。これによって、キャッシュ22内の或るキャッシュブロックがプロセサ12から要求されているアドレス範囲のキャッシュブロックに相当するか否かが判断される。ステップS3が“YES”であれば、ステップS5に進む。なお、このとき、有効ビットは真であることを要する。ステップS5において、インデックスカウンタ16で示されるディレクトリエンタリすなわちキャッシュブロックのダーティビットがクリアされ、ステップS7に進む。ステップS3が“NO”的ときは直接ステップS7に進む。ステップS7において、インデックスカウンタ16がインクリメントされ、ステップS9に進む。ステップS9では、インデックスカウンタ16の出力(addr-idx) & インデックスマスクレジスタ34aの出力(mask-idx) = インデックスマスクレジスタ34aの出力(mask-idx)であるか否かが判断される。これによってインデックスカウンタ16の上限を判断することができる。ステップS9が“NO”であればインデックスカウンタ16の出力はまだインデックスカウンタ16の上限ではないと判断され、上述の処理が繰り返される。ステップS9が“YES”であれば、インデックスカウンタ16の出力がインデックスカウンタ16の上限になったと判断され、終了する。

【0032】さらに具体的に説明する。ここで、キャッシュ22のブロックサイズをblocksizeとし、キャッシュ22のセット数をsetnumとする。また、addr, mask+1, blocksize, setnumは、それぞれ2のべき乗の値であり、定義より数1が成立する。

【0033】

【数1】 $addr\text{-tag} = addr / setnum / blocksize$

$addr\text{-idx} = (addr / blocksize) \% setnum$

$mask\text{-tag} = mask / setnum / blocksize$

$mask\text{-idx} = (mask / blocksize) \% setnum$

数1において、%はモジュロ演算を表している。ここで、キャッシュ22中に、block-tagなるタグを用いか

9

つblock-idx なるインデックスに対応するダーティなキャッシュブロックが存在したとする。このキャッシュブロックが対応付けられているメモリ領域 [block-min … block-max] は、数 2 によって表される。

【0034】

```
[数 2] block-min = (block-tag * setnum+block-id
x ) * blocksize
block-max = (block-tag * setnum+block-idx + 1)
*blocksize - 1
```

ここで、図 4 に示すアルゴリズムによって、[block-min … block-max] が [addr…addr+mask] に含まれるとき、また、そのときに限りこのキャッシュブロックのダーティビットがクリアされることを示す。

【0035】図 4 のアルゴリズムにおいて、ステップ S 3 を実行するときのインデックスカウンタ 16 の値 idx は、[addr-idx…addr-idx+mask-idx] の全ての値をとる。なぜなら、ステップ S 1 によって idx の初期値は addr-idx であり、制約条件より addr-idx & mask-idx = 0 であるから、ステップ S 7 にてインクリメントされた値 idx において、ステップ S 9 で idx & mask-idx = mask-idx が最初に成立するのは idx = addr-idx+mask-idx の場合となるためである。

【0036】すなわち、ステップ S 9 が “YES” になり終了するのは、idx = addr-idx+mask-idx の場合だからである。以下、mask+1 と setnum*blocksize の大小関係によって、mask+1 < setnum*blocksize の場合と、mask+1 ≥ setnum*blocksize の場合の 2通りに場合分けして考える。

【0037】(1) mask+1 < setnum*blocksize の場合
mask+1 < setnum*blocksize であるから、mask-tag = 0 が成立する。したがって、[block-min … block-max] が [addr…addr+mask] に含まれるということは block-idx が [addr-idx…addr-idx+mask-idx] に含まれ、かつ block-tag = addr-tag であるということに他ならない。したがって、図 4 に示すアルゴリズムにおいて、ステップ S 3 で検査されるキャッシュブロックは、その block-idx が [addr-idx…addr-idx+mask-idx] に含まれるもの全てであり、それ以外はない。また、mask-tag = 0 であるから、ステップ S 3 の検査は block-tag = addr-tag であるかどうかの検査と等価である。

【0038】(2) mask+1 ≥ setnum*blocksize の場合
この場合、addr-idx = 0 かつ mask-idx = setnum-1 が成立する。すなわち、図 4 のアルゴリズムにおいて、ステップ S 3 で検査されるキャッシュブロックはキャッシュ 22 中の全てのキャッシュブロックとなる。ここで、[block-min … block-max] が [addr…addr+mask] に含まれるということは block-tag が [addr-tag…addr-tag+mask-tag] に含まれるということに他ならない。なぜなら、block-idx は必ず [addr-idx…addr-idx+mask-idx] すなわち [0…setnum-1] に含まれるためであ

10

る。

【0039】ここで制約条件より、addr-tag & mask-tag = 0 であるから、ステップ S 3 で検査される block-tag & ~mask-tag = addr-tag であるかどうかは、block-tag が [addr-tag…addr-tag+mask-tag] に含まれるかどうかということと等価である。以上のように、図 4 に示すアルゴリズムでは、[block-min … block-max] が [addr…addr+mask] に含まれるキャッシュブロックについては必ずステップ S 5 が実行され、それ以外のキャッシュブロックについてはステップ S 5 は実行されないことがわかる。

【0040】この実施例によれば、たとえば、生存期間が終了した動的変数が置かれたダーティなキャッシュブロックがリプレースの対象となった場合に、そのキャッシュブロックのダーティビットをクリアすることによって不要な操作を防ぎ、計算機システム 10 の性能を向上させることができる。なお、上述の実施例では、clean-cache 命令に与えることのできるアドレス範囲に制限を設けていた。しかし、別の構成例として、より多くのハードウェア量を必要とするが、それぞれアドレス範囲の上限および下限と比較するための 2 個の大小比較器をタグ部の比較に用いることによって、任意のアドレス範囲を clean-cache 命令に与えることができる計算機システムを構成することもできる。

【0041】また、上述の実施例では、セットアソシティブ方式のキャッシュをとりあげて説明したが、ダイレクトマップ方式あるいはフルアソシティブ方式はセットアソシティブ方式の極端な場合であると捉えることができるため、ダイレクトマップ方式あるいはフルアソシティブ方式にこの発明を適用することもできる。

【0042】なお、clean-cache 命令は、上述のようにたとえば、ヒープ領域およびスタック領域に割り当てられた動的変数のメモリ割り当てを解放するときに、そのメモリ領域に対して用いる。特に、スタック領域に関しては、関数のリターン時に解放されるスタックフレーム領域全体に対する clean-cache を実行することができる。しかしながら、この発明は、プログラムによって利用される動的変数の生存期間が終了した場合、すなわち動的変数のメモリ割り当てを解放する場合以外でも用いられることはいうまでもない。

【図面の簡単な説明】

【図 1】この発明の一実施例を示すブロック図である。

【図 2】この実施例のタグマスクの構成の一例を示す図解図である。

【図 3】この実施例のインデックスマスクの構成の一例を示す図解図である。

【図 4】この実施例のキャッシュ制御部が clean-cache 命令を実行する場合のアルゴリズムを示すフロー図である。

【図 5】従来技術を示すブロック図である。

11

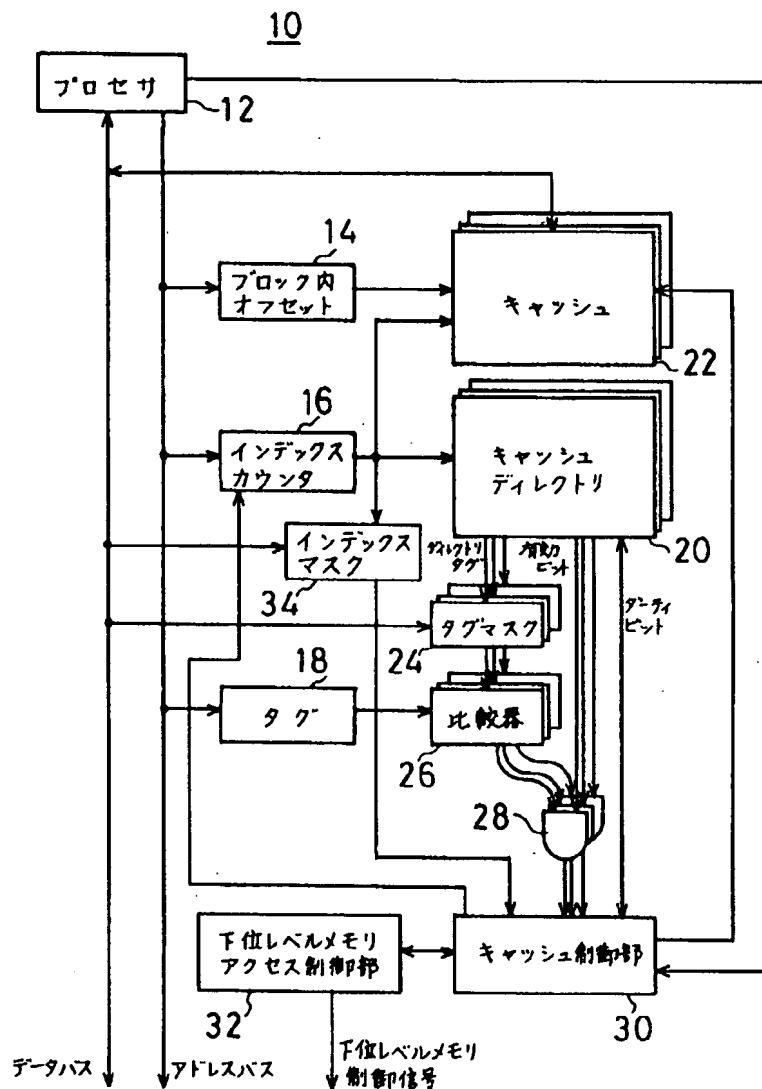
12

【符号の説明】

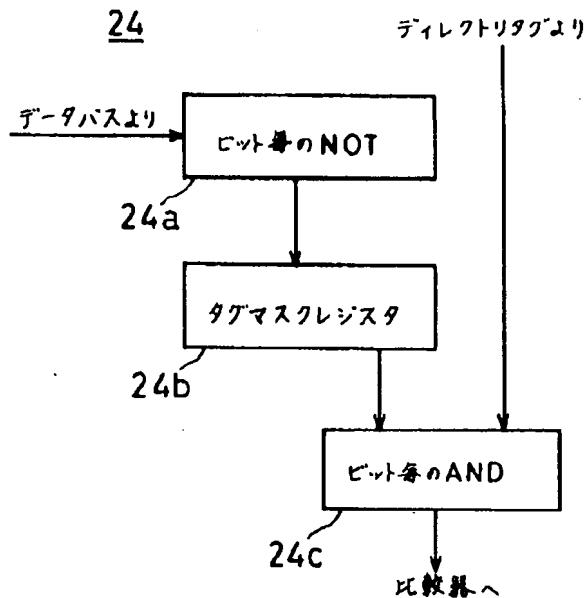
10 …計算機システム
 12 …プロセサ
 16 …インデックスカウンタ
 20 …キャッシュディレクトリ

22 …キャッシュ
 24 …タグマスク
 26 …比較器
 30 …キャッシュ制御部
 34 …インデックスマスク

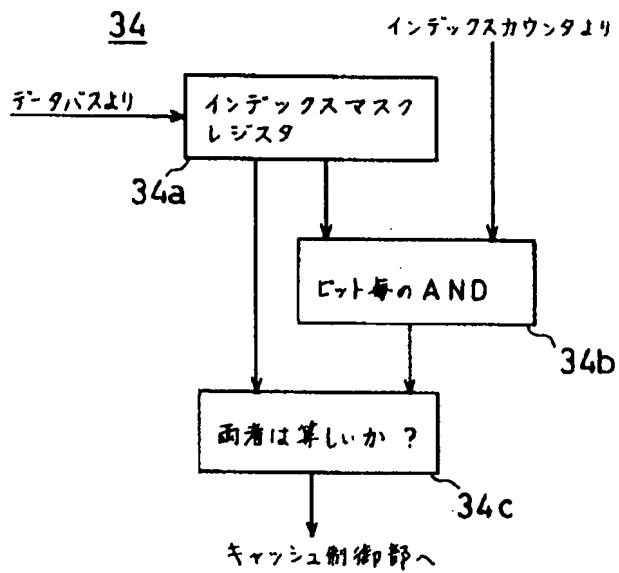
【図1】



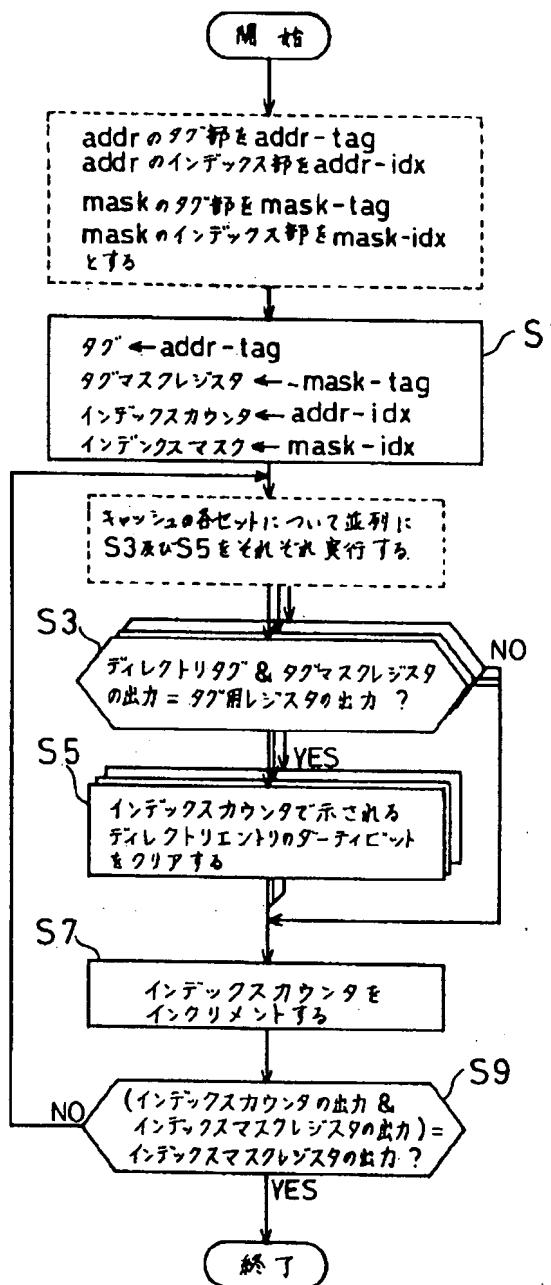
【図2】



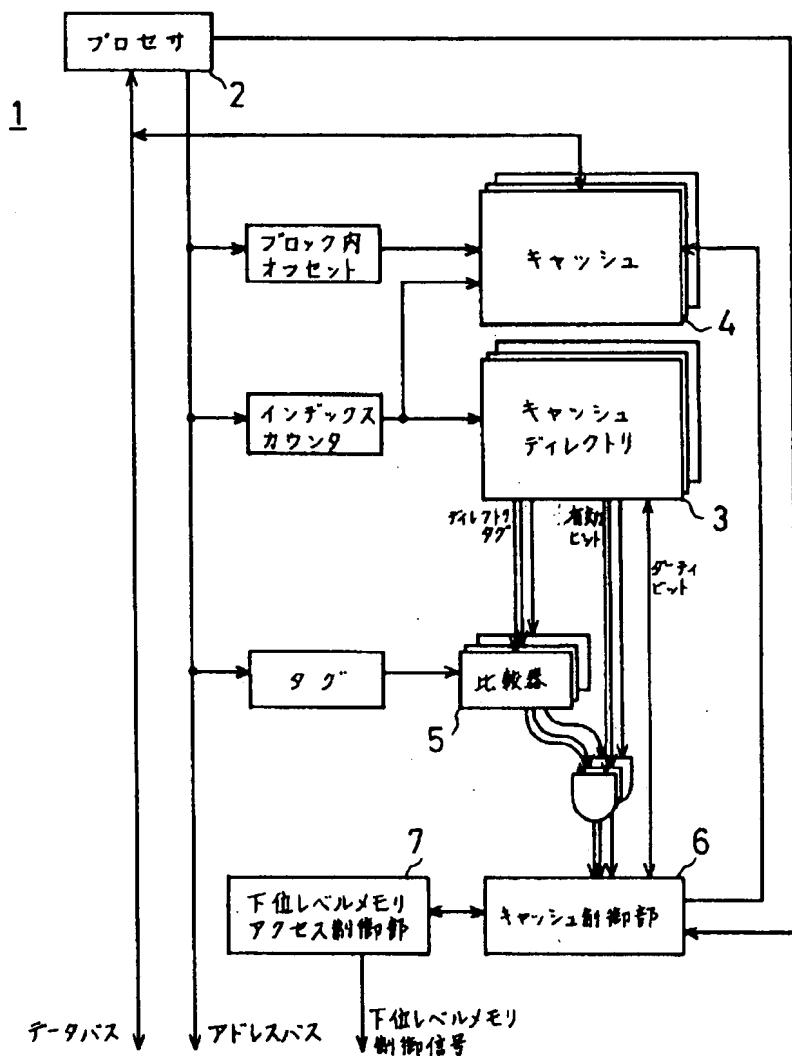
【図3】



【図4】



【図5】



PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-069417

(43)Date of publication of application : 12.03.1996

(51)Int.Cl.

G06F 12/12

(21)Application number : 06-203253

(71)Applicant : SANYO ELECTRIC CO LTD

(22)Date of filing : 29.08.1994

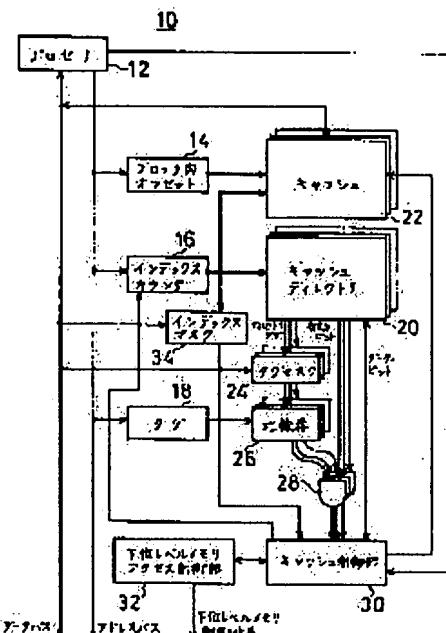
(72)Inventor : KOMURA YASUTO

(54) COMPUTER SYSTEM

(57)Abstract:

PURPOSE: To improve the performance off a computer system by preventing unnecessary writing to a memory block of low-order level.

CONSTITUTION: The cache block corresponding to an address range specified at a request of a processor 12 is detected by comparing a directory tag from a cache directory 20 with a tag from the processor 12 by a comparator 26. At this time, the cache block can efficiently be detected by masking the directory tag with a tag mask 22. Further, the value of an index outputted from the processor 12 is increased by an index counter 16 and the upper limit of the index counter 16 is detected with an index mask 34 according to the address range. The dirty bits of the cache block corresponding to the address range specified by the processor 12 are cleared until the upper limit of the index counter 16 is detected.



LEGAL STATUS

[Date of request for examination] 03.08.2001

[Date of sending the examiner's decision of rejection] 02.11.2004

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]